Proceedings of the 53rd Hawaii International Conference on System Sciences | 2020

Invasion of the Botnet Snatchers: A Case Study in Applied Malware Cyberdeception

Jared Chandler Kathleen Fisher
Tufts University
jared.chandler@tufts.edu
kfisher@eecs.tufts.edu

Erin Chapman Eric Davis Adam Wick
Galois, Inc.
{erin,ewdavis,awick}@galois.com

Abstract

In this paper, we provide the initial steps towards a botnet deception mechanism, which we call 2face. 2face provides deception capabilities in both directions – upward, to the command and control (CnC) server, and downward, towards the botnet nodes – to provide administrators with the tools they need to discover and eradicate an infestation within their network without alerting the botnet owner that they have been discovered. The key to 2face is a set of mechanisms for rapidly reverse engineering the protocols used within a botnet. The resulting protocol descriptions can then be used with the 2face network deception tool to generate high-quality deceptive messaging, against the attacker. As context for our work, we show how 2face can be used to help reverse engineer and then generate deceptive traffic for the Mirai protocol. We also discuss how this work could be extended to address future threats.

1. Introduction

Botnet detection is a popular field of inquiry, but often leaves open the question of what an administrator should do once they have discovered a botnet on their network. Eradication of a botnet can be difficult *a priori*, because bots can live in difficult-to-access devices on the network, or in devices whose function is critical enough that downtime is difficult to schedule. Indeed, it may do no good to eradicate a botnet from a device if the manufacturer has not yet (or never will) release a patch to protect it from re-infection in the future. Further, the mere act of attempting eradication sends a signal to the adversary that they have been discovered, causing them to adjust their botnet to thwart the administrator's mitigations.

We suggest using cyberdeception as an alternate or extension to direct botnet eradication: we first use human-supervised machine learning techniques to infer descriptions of the command and control (CnC) protocols at work. We then use the resulting protocol descriptions to synthesize highly realistic fake traffic. This fake traffic can be used for multiple deceptive purposes: to deceive the botnet controller that their infection is proceeding undetected and to trick botnet instances into exposing themselves to the administrator. In addition, the inferred information could be combined with honey data to help trace infections to their controllers.

In this paper, we use Mirai[1, 2] as a case study for early work in this effort. We provide a quick overview of Mirai in Section 3, but assume a basic understanding of its functioning and impact. Section 3 also sketches our goals and the threat model we adopt. In Section 4, we show how users can use 2face's inference assistance tooling to help understand the structure of Mirai CnC packets, to the point where they can generate a formal description of the packet syntax. We then discuss traffic generation in Section 5, and the operation and potential use of our 2face network deception system in Section 6. Finally, we discuss future work for 2face in Section 7 and conclude in Section 8. First, however, we provide an overview of existing work in the areas of deception and protocol reverse engineering in Section 2.

2. Related Work

The relatively wide adoption of network flow data [3] in large-scale networks has made it relatively easy for administrators to discover (either manually [4] or using machine learning [5]) anomalous flows within their networks. Often, however, it can be difficult to distinguish an anomalous flow caused by a user using a new application or device versus an anomalous flow caused by malware. Distinguishing between these two situations is particularly difficult in organizations that perform research and development tasks, such as government labs, large engineering organizations, and universities. In those organizations, identifying such flows requires an administrator with the time and professional network experience to hunt down possible sources for the anomalous flows[4]. While there have been some efforts to automate the process of sharing information amongst peers [6], such efforts

URI: https://hdl.handle.net/10125/63968
978-0-9981331-3-3

have not seen wide deployment. Another use for the work described in this paper would be to help system administrators understand what kinds of data are flowing across their network on suspicious channels, which may help them make decisions about the severity of threats more quickly.

The use of cyberdeception to detect adversaries within a network is not new [7], although much of this work focuses on heavyweight [8], virtual [9] or lightweight [10, 11] honeypots. Heavier-weight honeypots typically serve two roles: one in which they serve as detection systems on a network, and another in which they serve as an observational platform to understand the goals and techniques of an adversary. Some of the observations gained from such platforms have led to deeper manipulation of the adversary's view of a system or network, with research connecting deceptive techniques to attack graphs [12] or the use of inverted human factors engineering to distract attackers [13]. Several of these ideas have been connected to game-theoretic models of adversarial behavior to model how defenders might choose to deploy deceptions against an attacker to achieve a result [14]. The work in this paper extends this existing body of work by providing another mechanism that can be used to deceive an adversary by generating realistic traffic that can be used to inject false data into the system, by triggering malware in a controlled way, or by masking the remediation state post compromise. In particular, previous efforts in traffic generation (e.g., [15]) have been focused on generating traffic that can be used to effectively test and benchmark network equipment or networking algorithms. Our work is instead focused on generating traffic that is realistic to human operators, either directly or via their interactions with botnet nodes.

Protocol reverse engineering is broadly divided into two approaches [16]. One approach relies on interacting with a binary program capable of consuming or producing network messages. This binary program may be subject to methods derived from static analysis or interactive fuzzing to gain information about the message formats comprising the protocol and the state machine describing their use [17]. This approach requires the analyst to have some manner of interactive access to the binary program. An alternative approach relies only on captured data from a network trace [18, 19]. These efforts utilize a variety of information theoretic and machine learning techniques. A further distinction in protocol reverse engineering is made between human readable and binary format protocols. Human readable protocols are protocols which are transmitted as a string and are unambiguous in their interpretation at the character level. Binary format protocols transmit the compact binary representation of data and are ambiguous in their interpretation at the byte level. In this work, we adopt the second approach and assume the data is in binary, which is the most challenging combination.

3. Mirai & Experimental Setup

The Mirai botnet first appeared in 2016, and was characterized by rapid growth through infection of Internet-connected devices: digital video recorders, network attached video cameras, etc. At its peak the botnet exhibited a steady state of over 200,000 infected systems [20] and was used in several high-profile DDoS attacks. One of Mirai's more pernicious properties was the wide range of devices that it could affect, which made identifying all infections particularly time-consuming. One goal of our work is to facilitate the discovery of latent botnet infections by using deception to safely trigger them, thus disclosing their location.

The Mirai botnet is comprised of three types of systems: CnC servers which issue commands to bots; bots which attack victims and scan for new hosts to infect; and drop systems which perform infection of new, vulnerable hosts. The drop system uses brute force login to perform its attack, choosing from several known default user name and password combinations. Once infected, Mirai disguises its presence on a system by deleting its binary and using a pseudorandom string as its process name; it then fortifies the system by killing competing processes (other botnets, for example).

The Mirai botnet uses a binary format protocol for CnC communication [2]. This practice makes Mirai more difficult to reverse engineer from network traces than malware categories using human readable protocols such as XML or JSON. It was thus a particularly good target for 2face. Many traditional malware CnC systems use simple ASCII strings for control, which an administrator with a packet sniffer can easily reverse engineer.

3.1. Threat Model & Case Study Outline

In this case study we assume the following threat model. First we assume that a network we are responsible for has been infected by an unknown number of instances of botnet malware (for the purposes of this case study we use the Mirai botnet); the primary intended user of 2face is a network administrator at a university or other institution, managing a series of machines connected to the Internet via one or more firewalls. We assume that botnet instances are communicating with an outside CnC server under the command of an adversary who is actively using

the botnet for ongoing attacks. We assume that the adversary will alter their behavior and exact reprisals should they detect any effort to remove or interfere with the operation of the botnet. We further assume that we have identified an instance of a bot on our network, but that there may be others. In addition, we assume that it will take additional time to locate other bots, to develop patches for the infected machines, and to deploy these patches to the systems once located. We further assume that we are able to intercept the botnet CnC traffic on our network, but that we do not have access to the binary of the malware itself. In this case study, we do not address the mechanics of further system infection or scanning by bots.

For our protocol inference objective, we assume that we can intercept the traffic between the CnC server and some infected bots. Similarly, for our deception objective, we assume we have a firewall or router under our control between the CnC server and the infected hosts. Both of these objectives should be easily achieved by our target user – an academic, governmental, or corporate network administrator – through the injection of 2face at local network ingress/egress points.

Given these assumptions, our goal is to bring our system into a patched and uninfected state without arousing the adversary's suspicion. We broadly group our objectives into three phases:

- Phase 1: Reverse-engineer the CnC protocol from static network traces;
- Phase 2: Generate decoy systems which implement the Mirai CnC protocol; and
- Phase 3: Infiltrate and neutralize the botnet.

3.2. Generating Sample Data

Reverse engineering a network protocol from network traces assumes that we have a representative sample of messages in that protocol. The Mirai CnC protocol presents a substantial obstacle in this regard. First, these messages are not common, and they can only be collected on a path between a CnC server and a bot. Second, these messages reflect sensitive information that administrators may be loathe to release: the existence and address of the bot being the most critical. As such there are no public captures of Mirai CnC messages of more than a few messages.

To address this lack, we generated our own representative network traffic by creating a sandboxed Mirai Botnet comprised of a single CnC server and a single bot. We utilized publicly available source code for the various Mirai components and instantiated them as virtual machines connected on an isolated network segment from which we could record network traces. Mirai operators control a Mirai CnC server via a telnet interface through which they submit plain-text attack comments. The server subsequently disseminates binary versions of those commands to the connected bots. Each bot is connected to and takes orders from a single CnC server.

```
http 10.10.0.27 7 domain=ACM.ORG conns=77

http 10.10.0.27 10.10.1.7 7 domain=X.EDU
conns=600
```

Figure 1. Sample Mirai attack commands provided by the operator to the CnC server.

3.3. Traffic Generation

When a Mirai bot comes online, it initiates a connection to a CnC server automatically. After successfully creating a connection, the bot and the CnC server send heartbeat messages periodically. To capture a network trace of these protocol messages requires no action beyond running the CnC server and the bot.

To generate messages from the CnC server instructing the bot to attack a victim, we developed a script which connects to the telnet interface and issues plain-text attack commands. We parameterized our script by generating pseudo-realistic IP addresses as targets for the attack and domain names as parameters for the HTTP flood attack command we utilized. The CnC server distributes commands to the bots in a custom binary message format. It is this binary message format on which the bulk of our reverse engineering efforts were focused. We collected a network trace of these messages using an open source packet capture tool running on the same network as the bot and the CnC server.

4. Phase I: Automated Reverse-Engineering of Mirai CnC

Reverse engineering a network protocol from a network trace has two objectives. First, we recover the quantity and structure of discrete message formats for the protocol. Second, we infer a state machine describing the observed sequence of messages from various formats.

We use a human-machine centaur approach as the basis for these two objectives as illustrated in Figure 2. The centaur of Greek myth had human reasoning, but the strength and speed of a horse. In this domain, a centaur refers to a system designed to combine human reasoning with algorithms and computation such that when taken together, the combination exceeds what

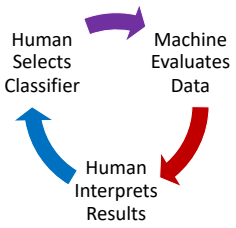

Figure 2. The 2face approach to reverse engineering.

either the human or the computer can do individually. A common example of a centaur system is active machine learning, in which a classifier can call out to a human expert for judgement when it needs additional information.

4.1. Reverse Engineering Message Formats

We conducted our case study reverse engineering the Mirai CnC protocol to make explicit the steps an analyst would perform to understand binary message formats. While previous work presents individual automatic techniques for reverse engineering from network traces [16, 18], there is a lack of step-by-step descriptions of the process an analyst would perform; we hypothesize that this lack stems from the unique nature of each reverse engineering task. We hope that by enumerating these steps, we can better understand where common reverse engineering tasks can be automated or improved. Previous work on reverse engineering the Mirai CnC protocol utilized the malware source code [2]. We utilize only network traces to investigate whether we can arrive at a similar format when source code or a binary is unavailable.

We define the successful reverse engineering of a message format as producing a description of the ordered field boundaries and their types (syntactic and semantic) that matches the original message format specification defined by the source generating the messages. We focus only on formats at the byte level as our base unit; formats that include fields that are not multiples of one byte are left for future work.

In this case study, we assume our sample contains only messages generated by the CnC protocol. We feel this assumption is reasonable given the wide availability of traffic filtering rules in packet capture tools and the fact that an administrator would likely limit their packet captures to a single (or small set) of UDP or TCP ports as part of their normal operation. We also assume that we are aware of some of the IP addresses at which the botnet attacks are directed. This assumption is consistent with a scenario in which a Mirai bot is discovered on a network and both the CnC traffic as well as the attack traffic generated by it are observed and captured as it passes through a switch or router. On the other hand, we do *not* assume that we know the plaintext commands that generated the binary packet data (*a la* Figure 1).

To reverse engineer message formats, the analyst first extracts the messages sampled from the network trace and performs a manual analysis of the distribution of message lengths by source. For example, the messages produced by a Mirai bot are of 1, 2 and 4 bytes in length, while the messages from the Mirai CnC server are either 2 bytes long or of variable lengths greater than 20 bytes. Based on observations of the data similar to our generated data, the analyst would observe the relative frequencies of message lengths, and form a reasonable hypothesis that the Mirai botnet protocol has some message formats which are fixed length and some message formats which are of variable length. They could then group messages by the combination of source (bot vs. CnC) and length, placing all messages greater than 20 bytes into a single "longer" message group. Here the analyst employs Shannon entropy to gain insight [21]. Shannon entropy can be described as a measure of diversity over a set of information [22]. It is a useful measure when reverse engineering as it does not make any assumption about the semantic characteristics of the data, only the quantity of information, and is a direct measure of how well one model (or subset of messages) predicts another. By calculating the Shannon entropy and number of distinct values for each group, the analyst would discover that the groups of messages of 1, 2, and 4 bytes in length all have both a single distinct value and a Shannon entropy of 0; these facts indicate they are constant within the group, and we can begin to generate theories that these messages are heartbeats and acknowledgements where the existence and length of the message encode all the necessary information. For each group with a single distinct value, the analyst would likely perform no further analysis, under the assumption that the message format and the value are equivalent. Note that a limitation of reverse engineering is that a format can only be inferred with respect to the sample captured by the network trace. A different or larger sample could contain information which would result in inferring a different format.

4.2. Attack Message Format Inference

Having reverse-engineered the simple message formats, the analyst can next concentrate on the remaining group of messages: the ones that are larger and of varying length. Figure 3 provides examples of simplified Mirai attack command binary messages from Figure 1 and summarizes how toolinng supports the steps of our methodology. In general, our tools and methods are designed to come up with reasonable theories regarding the structure of a message, and then

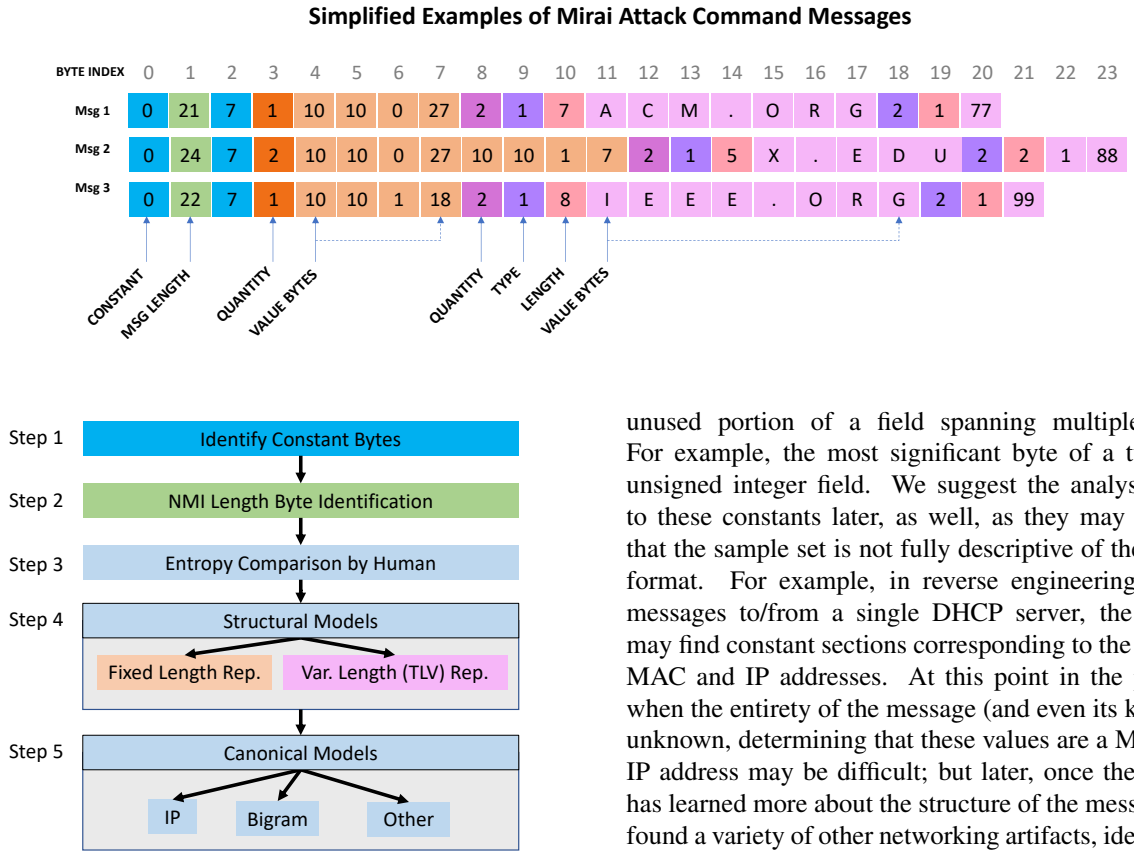

Figure 3. Sample Mirai attack commands and the corresponding message format inference methodology. Message bytes are color-coded by corresponding inference step.

validate that each candidate structure is valid against all incoming data. If, for example, a proposed definition of the message leads to lengths before or after the actual end of a message, or includes a checksum that fails on some of the input data, then we dismiss that candidate definition and seek another explanation.[1]

Step 1. Our approach first helps the analyst extract a structural description of the message format before trying to identify individual field boundaries and semantic types. In this stage, the analyst first uses 2face's automated tooling to discover byte positions relative to the message start where the corresponding payload takes on constant values across all messages. In particular, we pay attention to byte positions which contain the value 0, as these positions are likely an unused portion of a field spanning multiple bytes. For example, the most significant byte of a two-byte unsigned integer field. We suggest the analyst return to these constants later, as well, as they may indicate that the sample set is not fully descriptive of the packet format. For example, in reverse engineering DHCP messages to/from a single DHCP server, the analyst may find constant sections corresponding to the server's MAC and IP addresses. At this point in the process, when the entirety of the message (and even its kind) are unknown, determining that these values are a MAC and IP address may be difficult; but later, once the analyst has learned more about the structure of the message and found a variety of other networking artifacts, identifying such constants will likely be easier.

Step 2. As the messages for Mirai CnC are of varying length, the analyst's next goal will be to seek regions in each message that share high normalized mutual information (NMI) [23, 24, 25] with the length of the message. NMI describes the mutual dependence between two variables, and thus the amount of information that can be obtained about one such variable by observing the other. Since lengths can vary, as well as byte ordering, we normally suggest starting from small, unsigned, network-order numeric types and then expanding from there when trying to find such regions. For instance, in Mirai, we would suggest starting by calculating the NMI for each unsigned byte in the message; this calculation is an example of leveraging the machine portion of the centaur approach.

In Mirai, the analyst uses 2face's automated tooling to find that the byte values at index 1 shows NMI of 1.0, indicating high explanatory power for the total length of the message. We note that the byte at index 0 is a constant value of 0 across all messages. We might then infer that together these two bytes form a length field. We note that leading with the length is very consistent with other protocol message formats where the total length of the message is present early in the message to

[1] One issue that we do not address in this work is the issue of dirty data; we assume, for this section, that all of the input data is valid and unmodified. In the case of errors in the analyzed data stream, then we change our heuristic to check for consistency with known error rates for the underlying channel.

allow a receiving system to know when to stop reading bytes for a discrete message unit.

Step 3. The analyst then calculates the entropy at a particular (k) position relative to the message start for each message (m) in the sample (M). This calculation is defined as $-\Sigma_{i \in M_k} P_i log P_i$ where M_k is the bag formed by the kth byte of each message m in the sample M, and P_i is the probability of value i in M_k.

The analyst uses automated tooling which first calculates the entropy of the values of the first byte of each message. This value is compared with the entropy of the values of the second byte of each message. Continuing in this manner allows the comparison of the amount of information contained at each byte position relative to the start of the messages. Through numerical comparison and visualization similar to the approach described in Kleber [26], the analyst will note regions of differing entropy. From this pattern, they might hypothesize that the messages are comprised of a mixture of different repeated field formats. If it were a single field format, for example, they would expect repeated regions to have consistent entropy. The analyst's next goal would then be to determine what kind of repeated format they are dealing with. If there is a consistent step size between discrete message lengths, for example, all the message lengths could be described as a header of size n plus some number of s-byte chunks, and thus all the messages had a size of $x * s + n$, then the analyst could assume that there was a single repeated field with a width of the step size s. In this case there is not, but we have found this pattern to be common in other protocols, and a relatively easy property to check for at this stage.

Step 4. Discovering the syntax of repeated fields is one of the more difficult tasks for the analyst. Other work on protocol reverse engineering has relied on string alignment algorithms from bioinformatics to address this problem [27, 28, 29]. We instead use a "guess, then validate" approach, by drawing candidate hypotheses from common data serialization techniques, deserializing the sample data accordingly, and then determining whether or not the result passes simple sanity checks.

Prominent among these techniques are length value (LV) encoding and type length value (TLV) encoding. These encodings are used in the ASN.1 BER [30] specification and the Google Protocol Buffers (ProtoBuf) library [31].

With TLV encoding a field is first encoded with its field type, followed by a description of the field length, followed by its value as consecutive bytes. This encoding is popular with designers because it affords a certain level of forward- and backwards-compatibility, as implementations that are not capable of parsing a certain type value can easily jump past it using the length information. In other cases, some portion of the repetition (either the number of repetitions or the size of the structure being replicated) is fixed, and as a result less information need be provided: a count field suggesting the number of elements, for example, or just a type value for each repetition.

Mirai attack messages include two examples of repeated fields, as our analyst will shortly discover. First, since TLV encoding is so common, the analyst performs a brute force search across the captured pattern data, looking for possible sections of the data that match that pattern. To do so, the brute force algorithm starts at a given offset of the message, and then attempts to overlay a TLV encoding at a variety of normal sizes (1 byte type / 1 byte length, 2/2, 1/2, etc.). For each of the assumed encodings, we can then determine if the encoding makes sense by seeing how each assumed encoding aligns with the end of the packet data; in cases in which a boundary consistently occurs at the end of the packet, when computed across all of our captured packet data, the encoding should be considered a candidate by the analyst.[2] At this point, a certain level of interaction with the human is required, as intuition is likely to be the best guide as to whether one TLV candidate makes more or less sense than its peers.

For Mirai, it becomes clear that there is likely a TLV encoding at the end of a message. With this information, the analyst now holds a hypothesis that the Mirai message contains, in order: a 2-byte length field, a 4-byte fixed-width field, a section of unknown bytes, and a set of TLV-encoded repetitions. A quick check, similar to the length check described earlier, can easily determine that the last of the unknown bytes exactly corresponds to the number of TLV-encoded items. By applying a process similar to the automated search used for TLV-encoded data, the analyst discovers the unknown bytes consist of 5-byte fixed length repetitions with a quantity given by the first unknown byte. At this point, the analyst has completely parsed the structure of the data, and can switch to analyzing the frequency of the values contained within this structure.

Step 5. In this particular case, the fixed 5-byte quantity is consistent with 5-byte CIDR IP address ranges [32]; contextual clues, like NetFlow records associated with attacks seen from compromised nodes, would help in this determination. We also note that the subsequent variable length repetition contains two types of values by decoding them as TLV groups. One

[2] Similarly, if the encoding ends at a consistent point before the end of the packet, this may indicate that we have correctly identified a TLV structure, but that there is additionally a trailer structure in the syntax.

group is uniform in length, and is a numeric parameter. The other is varying in length, but consistent with bi-gram probabilities [21, 33] associated with an ASCII representation of English text. Manual examination is used to determine that the values of this group are ASCII domain names.

We note that, for many fields, there are other heuristics that we can use to aid the analyst in generating strong guesses regarding the semantic content of a field. In this case, we have used bi-gram probabilities that heuristically suggests ASCII data, which the analyst then validated. Similar heuristics exist for determining, for example, time stamps in various resolutions, positional coordinates, angles, IP addresses/netmasks, etc. One aspect of our future work is to build a library of these distributions, which can be automatically applied to the data.

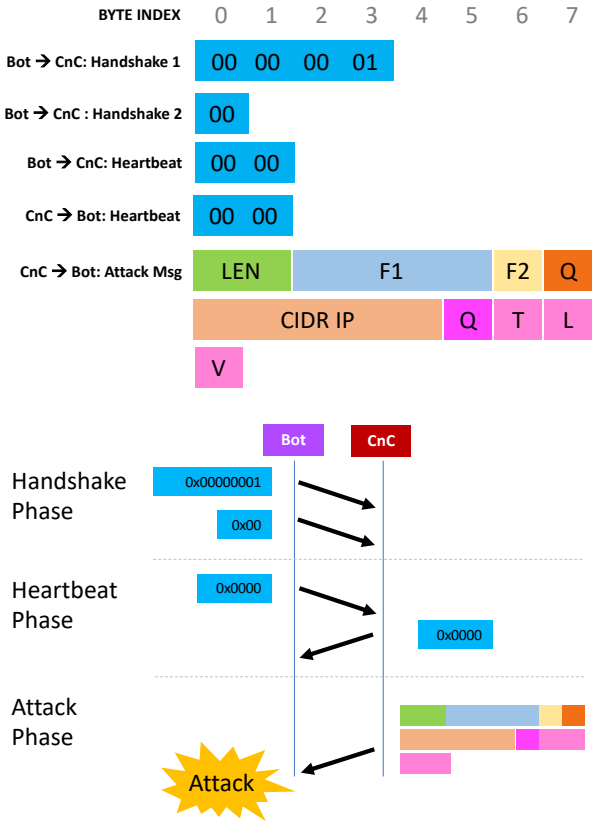

Figure 4. The format and sequence of messages in the Mirai state machine.

4.3. Reverse engineering state machines

Having developed message format descriptions for each type of CnC message observed in the network trace, the analyst now turns their attention to developing state machines which describe the exchange of message between bot and CnC server. In general, the problem of automaton inference can be difficult and is well studied [34, 35]. However, Mirai, like many botnet protocols, exhibits a very simple sequence of message exchanges that is readily apparent through manual examination. Specifically, the Mirai bot first sends a pair of handshake messages, followed by a heartbeat message approximately 60 seconds later. The Mirai CnC server responds to the heartbeat message with a reply. Heartbeat messages are exchanged in this fashion every 60 seconds. The Mirai CnC server can at any point after the handshake send an attack message to the bot initiating an attack. The discovered message formats and sequencing of message types are shown in Figure 4.

5. Phase II: Synthesizing Realistic Mirai CnC Traffic and Behaviors

In the second phase of our case study we show how an analyst can create decoy systems that emulate the traffic and behavior of real Mirai bots and CnC servers.

Of the protocol message formats inferred in the previous phase, only the attack message exhibits variation. The handshake, heartbeat and heartbeat reply are all statically-valued messages. For our goal of maintaining the illusion of a functioning Mirai botnet, it is sufficient to simply identify an attack message rather than synthesize one. That being said, Mirai is a particularly pernicious bot, so generating attack commands may also be useful to flush out latent bots in the network. Specifically, it may be useful to send attack commands to all potentially-infected devices – targeting an administrator-controlled, sacrificial node – to trigger latent botnets into exposing themselves.

5.1. Decoy CnC Server

We implement a decoy CnC server which has the following three operations: accept handshake messages, accept heartbeat message, and reply to heartbeat message. The decoy CnC server has two modes: cold and warm. In the cold mode, the decoy CnC server only performs the accept handshake message operation. Once this task has been completed the decoy transitions to the warm mode, where it will accept and reply to heartbeat messages. These two modes allow a CnC decoy to be created in whichever state would match what a specific real Mirai bot expects.

Given the understanding of the packet formats developed in the previous section, creating this decoy server is a relatively trivial development exercise. We will discuss future integration plans that make this exercise even easier in Section 7.

5.2. Decoy Bot

Similar to the decoy CnC server, we implement a decoy Mirai bot with four operations: generate handshake messages, generate heartbeat message, accept heartbeat message reply, and finally, accept attack message. The decoy bot has two modes, cold and warm, mirroring those of the decoy CnC server. In the cold mode, the decoy bot will generate handshake messages and then transition to the warm mode, where it will generate heartbeat messages and accept replies every 60 seconds. The decoy bot will also accept an attack message at any time in either mode, but takes no action in response.

6. Phase III: Botnet Deception

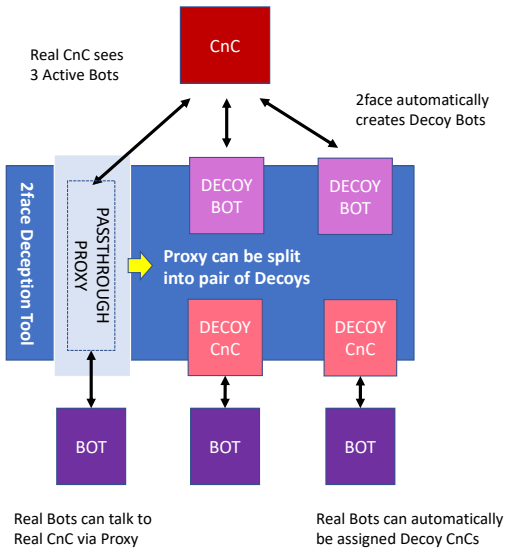

Figure 5. The organization of the 2face network deception tool.

The final phase of our case study tackles deception of the adversary controlling a botnet. It is important to note that the goal of this effort is to delay the adversary's awareness of botnet mitigation for as long as possible. That being said, the owner of a Mirai network, for example, may become aware that their botnet has been corrupted should they launch an attack against a target. At this point, if the botnet is mostly (but not entirely) mitigated, they may (rightly) conclude that their botnet has been discovered because of the unexpectedly low resulting attack volume. Still, any confusion we can create through having their botnet appearing to be alive by sending heartbeats can only be positive.

With this goal in mind, we have created the 2face network deception tool. The 2face network deception tool is designed to facilitate deception operations against an adversary in control of a botnet on our network. We illustrate the organization of the tool with regard to these capabilities in Figure 5. A network operator controls the 2face network deception tool in real time through a telnet interface, or they can observe the state of the ongoing deception operation through a visual interface. The key requirements for the tool are as follows:

1. The ability to act as a transparent proxy between a real Mirai bot and a real Mirai CnC server.
2. The ability to transition from using a passthrough proxy (shown on the left of Figure 5) for a particular CnC / bot connection, to using a twinned pair of decoys. As a result of this transition, the CnC will now be talking to a decoy bot under 2face's control, and the real bot will be talking a decoy CnC. This is shown in Figure 5 in the center section.
3. The ability to create rules which govern the behavior of existing or newly discovered Mirai bots when they communicate with the 2face network deception tool. This allows us to detect and manage new infections as they try to reach the CnC, and allows us to provide the illusion of a steady growth rate to the CnC during mitigation.

The first step in the deceptive use of the 2face network deception tool is to install it as a proxy at relevant connection points: upstream connections to the ISP, intersections between major network segments, etc. In its initial use, the tool serves as a transparent proxy for traffic routed to it. Because the analyst has already determined the structure of the underlying data, they can initially use this proxy to simply gather statistics about the botnet's current presence on the network, as well as patterns of attacks. As an example, for Mirai, which signals back to the CnC quite often, the operator can quickly gather a good estimate on the number of devices infected on their network as well as the growth rate of the infection; this information can be useful later in the deception campaign. Additionally, information about the targets for attacks can provide insight into who the adversary is and the reason or goals for the attacks being launched.

Once botnet mitigation begins, connections between the CnC network and each individual bot can be severed, with the upstream (to CnC) link transferred to a decoy CnC, and the downstream (from CnC) link transferred to a decoy bot; we show this pattern in Figure 5. This severing has the immediate effect of disabling the adversary from launching attacks using the bot, although it does not remove the underlying infection.[3] To the botnet operator, the quantity of bots appears unchanged.

[3] One concern with more advanced bot software is that individual bots might use a loss of signal from the CnC node as a signal to shift

During the mitigation process, decoy CnCs can be shut down as their associated bot is disabled. At this point, the administrator may wish to use any information gathered about the botnet growth rate to provide a consistent vision to the adversary through the appropriate creation of additional decoy bots. For more quiet botnets, in which there is not a consistent heartbeat message sent to the CnC server, administrators can set up a sacrificial server and then use the 2face network deception tool to generate attack messages targeting it. These messages can then be broadcast across the network with the goal of causing any latent infected nodes to break cover and attack the sacrificial machine.

We imagine the 2face network deception tool continuing to operate until the botnet infection has been cleaned. In many cases, it may be useful to leave the tool running until it is clear that the deception has been discovered, as it is possible that previously-clean systems may be infected or reinfected over time.

7. Discussion & Future Work

A critical factor in the 2face network deception tool is its ability to generate deceptive network traffic that is indistinguishable from real network traffic. Our technique of using human-machine teaming to discover the structure of network traffic aids in this, as the workflow we have suggested includes several opportunities to cross-check decisions against gathered network data. In the context of a real botnet, new traffic should also be generated and collected at regular intervals, providing even more testing and training data. Once the format has been inferred, we can guarantee that generated traffic conforms to the computed format and varies according to observed probability distributions.

While we are confident in the syntactic correctness of our approach, several concerns for a 2face-like system remain. For example, we assume we have access to the plaintext structure of all messages, either because the adversary does not use encryption (the common case in botnet engineering), or uses encryption incorrectly in a way that allows easy decryption. In addition, several aspects of this case study involved manual tooling to go from an inferred format to a message generation mechanism. We have experimented using the PADS data description language [36] to automate this process, integrating an ability to specify generation routines within a PADS description. In future work, we plan to integrate this generation mechanism more fully, and then integrate the whole framework into an existing network cyberdeception tool. [37]

into a different phase of operation. The use of the 2face network deception tool can help mitigate this risk by deceiving the bot into believing that its CnC system is still alive and aware.

On the analysis side, we have found that the tools described in this paper are sufficient for understanding and analyzing relatively straightforward packet formats that use common networking techniques such as TLV encoding. One area in which our current system is lacking is a library for detecting common data distributions associated with various data types: distributions for time values, angle values, positions, common private IP subnets, etc. Such recognizers will be useful to analysts *a priori*, but are also likely to be useful as part of a feedback loop in working through various hypotheses associated with packet data.

8. Conclusion

In this paper we have presented 2face, a set of tools for helping discover botnet infections within a network and for fooling the adversary into believing a cleaned network is still compromised. The key capabilities of 2face are its ability to help an administrator quickly decipher CnC protocols, including previously-unknown protocols. Once the structure has been identified, we have shown how the 2face network deception tool can be used to hide the administrator's discovery of the problem for as long as possible, and to perform controlled triggering of latent infections within the network. As an example, we have shown how we can use 2face's automated protocol reverse engineering tooling to decipher the Mirai protocol, and then generate deceptive traffic based on this information.

While we believe 2face shows great promise, we note that our work opens up questions regarding more complicated interactions. For example, we have not directly addressed the use of more complex protocols, or protocols that use contextual information (timing, for example) as part of their structure. One critical need is the ability to better judge what forms of information could be used to distinguish real from deceptive traffic, and discover metrics for determining the effectiveness of the latter. Still, for simple CnC algorithms, 2face represents an interesting new capability for system administrators to use in decreasing the threat of botnet infections in their networks.

References

[1] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, *et al.*, "Understanding the mirai botnet," in *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pp. 1093–1110, 2017.

[2] C. McAuley, "Mirai: A botnet of things," Oct 2016.

[3] R. Hofstede, P. eleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras, "Flow monitoring explained: From packet capture to data analysis with netflow and

ipfix," *IEEE Communications Surveys Tutorials*, vol. 16, pp. 2037–2064, Fourthquarter 2014.

[4] J. Jones, "Finding the needle in the haystack." FloCon, 2017.

[5] V. Sekar, N. Duffield, O. Spatscheck, J. E. van der Merwe, and H. Zhang, "Lads: large-scale automated ddos detection system," in *Proceedings of the 2006 USENIX Annual Technical Conference*, pp. 171–184, 01 2006.

[6] J. Berkes, "Ddos defense with a community of peers (3dcop)." FloCon, 2017.

[7] F. Cohen, "A note on the role of deception in information protection," *Computers & Security*, vol. 17, pp. 483–506, 12 1998.

[8] L. Spitzner, *Honeypots: Tracking Hackers*. Addison-Wesley Professional, 2002.

[9] N. Provos and T. Holz, *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*. Addison-Wesley Professional, 2007.

[10] N. Provos, "A virtual honeypot framework," in *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, (Berkeley, CA, USA), pp. 1–1, USENIX Association, 2004.

[11] A. Wick, "Selling unikernels: The cyberchaff story." QCon San Francisco, 2017.

[12] F. Cohen and D. Koike, "Feature: Leading attackers through attack graphs with deceptions," *Comput. Secur.*, vol. 22, pp. 402–411, July 2003.

[13] R. Gutzwiller, K. Ferguson-Walter, S. Fugate, and A. Rogers, "oh, look, a butterfly!" a framework for distracting attackers to improve cyber defense.," in *Proceedings of the Human Factors and Ergonomics Society*, 10 2018.

[14] K. Ferguson-Walter, S. Fugate, J. Mauger, and M. Major, "Game theory for adaptive defensive cyber deception," in *Hot Topics in the Science of Security Symposium*, HotSoS '19, 04 2019.

[15] K. V. Vishwanath and A. Vahdat, "Realistic and responsive network traffic generation," in *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '06, (New York, NY, USA), pp. 111–122, ACM, 2006.

[16] S. Kleber, L. Maile, and F. Kargl, "Survey of protocol reverse engineering algorithms: Decomposition of tools for static traffic analysis," *IEEE Communications Surveys & Tutorials*, vol. 2018, 2018.

[17] C. Y. Cho, E. C. R. Shin, D. Song, *et al.*, "Inference and analysis of formal models of botnet command and control protocols," in *Proceedings of the 17th ACM conference on Computer and communications security*, pp. 426–439, ACM, 2010.

[18] W. Cui, J. Kannan, and H. J. Wang, "Discoverer: Automatic protocol reverse engineering from network traces.," in *USENIX Security Symposium*, pp. 1–14, 2007.

[19] G. Bossert, F. Guihéry, and G. Hiet, "Towards automated protocol reverse engineering using semantic information," in *Proceedings of the 9th ACM symposium on Information, computer and communications security*, pp. 51–62, ACM, 2014.

[20] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, *et al.*, "Understanding the mirai botnet," in *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pp. 1093–1110, 2017.

[21] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, pp. 379–423, July 1948.

[22] T. M. Cover and J. A. Thomas, *Elements of information theory*. John Wiley & Sons, 2012.

[23] W. Li, "Mutual information functions versus correlation functions," *Journal of statistical physics*, vol. 60, no. 5-6, pp. 823–837, 1990.

[24] D. H. Wolpert and D. R. Wolf, "Estimating functions of probability distributions from a finite set of samples," *Physical Review E*, vol. 52, no. 6, p. 6841, 1995.

[25] Wikipedia contributors, "Mutual information — Wikipedia, the free encyclopedia," 2019. [Online; accessed 10-June-2019].

[26] S. Kleber, H. Kopp, and F. Kargl, "{NEMESYS}: Network message syntax reverse engineering by analysis of the intrinsic structure of individual messages," in *12th {USENIX} Workshop on Offensive Technologies ({WOOT} 18)*, 2018.

[27] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of molecular biology*, vol. 48, no. 3, pp. 443–453, 1970.

[28] W. Cui, V. Paxson, N. Weaver, and R. H. Katz, "Protocol-independent adaptive replay of application dialog.," in *NDSS*, Citeseer, 2006.

[29] G. Wondracek, P. M. Comparetti, C. Kruegel, E. Kirda, and S. S. S. Anna, "Automatic network protocol analysis.," in *NDSS*, vol. 8, pp. 1–14, 2008.

[30] I. Recommendation, "690 (2008)— iso/iec 8825-1: 2008," *Information technology–ASN. 1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*.

[31] "Google Protocol Buffers: Googles Data Interchange Format." `https://code.google.com/p/protobuf/`. Accessed: 2019-08-27.

[32] Wikipedia contributors, "Classless inter-domain routing — Wikipedia, the free encyclopedia," 2019. [Online; accessed 11-June-2019].

[33] J. H. Martin and D. Jurafsky, *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. Pearson/Prentice Hall Upper Saddle River, 2009.

[34] D. Angluin, "Learning regular sets from queries and counterexamples," *Information and computation*, vol. 75, no. 2, pp. 87–106, 1987.

[35] C. De la Higuera, *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010.

[36] K. Fisher and R. Gruber, "Pads: A domain-specific language for processing ad hoc data," in *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '05, (New York, NY, USA), pp. 295–304, ACM, 2005.

[37] A. Wick, "I want your flows to be lies." FloCon, 2017.